Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 47 (2004) 1363–1378
An International Journal computers & mathematics with applications
www.elsevier.com/locate/camwa

Collision Detection Algorithm of a Continuous Type Using Spherical Extreme Vertex Diagrams

HYOUNG SEOK KIM
Department of Multimedia Engineering, Dongeui University
San 21, Gaya-Dong, Pusanjin-Gu, Pusan, 614-714, Korea
hskim@dongeui.ac.kr

HONG OH KIM
Department of Mathematics, KAIST
373-1, Kusong-Dong, Yusong-Gu, Taejon, 305-701, Korea

SUNG YONG SHIN
Department of Computer Science, KAIST
373-1, Kusong-Dong, Yusong-Gu, Taejon, 305-701, Korea

(Received July 2001; revised and accepted March 2003)

Abstract—Most collision detection algorithms are ones of a discrete type which utilize convexity and local coherence to verify the closest points. However, the incremental algorithms have such critical drawbacks like failure to detect high-speed collisions and dependence on the time complexity on the time step size of animation. In this paper, in order to find a new method which can resolve the drawbacks, we are concerned with a more restricted but nontrivial version of the problem: given a fixed infinite plane H and a moving convex polyhedron P, compute their collision time. Our algorithm utilizes the spherical extreme vertex diagram that is the embedding of the dual graph P onto the Gauss sphere. Exploiting the diagram, we are able to efficiently compute the sequence of the extreme vertices v_is, $i = 1, \ldots, m$ and find the time interval $[t_i, t_{i+1}]$ during which v_i is the extreme vertex of P. With the sequences of the consecutive time intervals and extreme vertices, we construct a distance function between P and H. Hence, we can compute their collision time within a given tolerance by applying the interval Newton method to the distance function. Moreover, the total time complexity is independent of the time step size of animation.

Keywords—Extreme vertex problem, Convex hull, Spherical Voronoi diagram.

1. INTRODUCTION

Collision detection problems have an extensive background in the fields of computational geometry and robotics [1–5]. The efficient evaluation of the Euclidean distance between two objects plays an important role in computing their exact collision time. In the field of computational geometry, the Euclidean distance problem from a 3D convex polyhedron P to an infinite plane H

This work was supported by Korea Research Foundation Grant (KRF-99-003-E00395).

0898-1221/04/$ - see front matter Typeset by AMS-TEX
doi:10.1016/j.camwa.2004.04.020

is reduced to an extreme vertex problem, i.e., how to identify a vertex of P that first touches H when H is translated toward P.

Edelsbrunner [6] was able to solve this problem in $O(\log n)$ time after $O(n)$ time and space preprocessing, where n is the number of vertices in P. The efficiency of his method is due to an elegant data structure representing a hierarchy of convex polyhedra. Although the nested polyhedral hierarchy gives an optimal solution for the static case, it is not efficient to directly apply the hierarchy to a more generalized version of the extreme vertex problem with the object in motion, which is to find out the sequence of the extreme vertices during a given animation time interval. The total time complexity of this method is dependent on the time step size of animation because the number of evaluation of the Euclidean distance is inversely proportional to the time step size. Moreover, this method cannot generate the distance function between the polyhedron and the plane, which will play an important role in our algorithm of a continuous type.

The more efficient solutions of the generalized version may be an important tool of a collision detection problem between the convex polyhedron and the plane. The popular algorithms for collision detection problems are the incremental ones which utilize convexity and local coherence to verify the closest features (vertices, edges, or faces) [2,7–9]. The Euclidean distance of the next time instance $t_0 + \Delta t$ may be computed by the local test for the neighboring features of the closest feature at the time instance t_0. However, the incremental algorithms have such critical drawbacks like failure to detect high-speed collisions and dependence on the total time complexity on the time step size of animation [10]. For example, we consider an environment with a moving polyhedron bouncing off the ground. If the polyhedron collides with the ground at a time instance t_0, then the torque of the polyhedron changes and thus its angular speed may abruptly increase according to its collision feature. In this case, the real closest feature of the next time instance $t_0 + \Delta t$ can appear in the farthest part from the collision feature, but the incremental algorithm computes the closest feature from the neighbor of the collision feature to all features. Hence, the time complexity in this case may be $O(n^2)$. Therefore, in order to apply the local test while keeping this incremental strategy, the time step size has to be decreased up to an appropriate size.

In this paper, we give a more efficient solution of the generalized version of the extreme vertex problem with the object in rigid motion, and then compute their collision time within a given tolerance ϵ. This approach is called a continuous type because the total time complexity of our algorithm is independent of the time step size of animation. We employ the notion of the Gauss sphere [11] to construct spherical extreme vertex diagrams for solving both the original problem and its generalized version efficiently. Using the spherical extreme vertex diagram [12–15], we give a new algorithm for computing the Euclidean distance from a polyhedron to a fixed infinite plane in $O(\log n)$ time after $O(n)$ time and space preprocessing. With this algorithm as a tool, we are able to compute the sequence of the extreme vertices v_is, $i = 1, \ldots, m$, of P with respect to H and thus find out the time interval $[t_i, t_{i+1}]$ during which v_i for each $i = 1, \ldots, m$ is the extreme vertex of P. That is, for every time instance in $[t_i, t_{i+1}]$, we can compute the Euclidean distance in a constant time. Hence, our algorithm is more efficient than Edelsbrunner's method. With the sequence of the consecutive time intervals, we construct a distance function of time between them. The polyhedron collides with the plane where the function first vanishes. In order to compute the collision time, we apply the interval Newton method [16] to the distance function. Hence, we can compute the collision time within a given tolerance ϵ. Moreover, the time complexity of our collision detection algorithm is independent of the time step size of animation.

The rest of this paper is organized in the following manner. In Section 2, we show that the extreme vertex problem can be transformed to a spherical point location problem on the Gauss sphere by employing the dual of a convex polyhedron and its projection on the sphere. It is the basic idea of our algorithm. In Section 3, a brief review of quaternion curves is presented and their corresponding spherical paths are generated on the Gauss sphere. In Section 4, we

present how to construct the distance function between a polyhedron and a plane by using the spherical extreme vertex diagram. In Section 5, we show that the collision detection problem can be solved by applying the interval Newton method to the distance function. Experimental results are shown in Section 6. Finally, we conclude the paper with some remarks in Section 7.

2. EXTREME VERTEX PROBLEM

Let P be a convex polyhedron which is an intersection of n half-spaces. Let the boundary of P consist of n convex faces, F_i, $i = 1, 2, \ldots, n$. We denote the outward unit normal vector of F_i by $\mathbf{n}F_i$ for each i. Suppose that P is sufficiently apart from a fixed infinite plane H so that P always lies above H, which makes sense if we regard H as the surface of the ground. The unit normal vector of H pointing below is denoted by $\mathbf{n}H$ (see Figure 1).

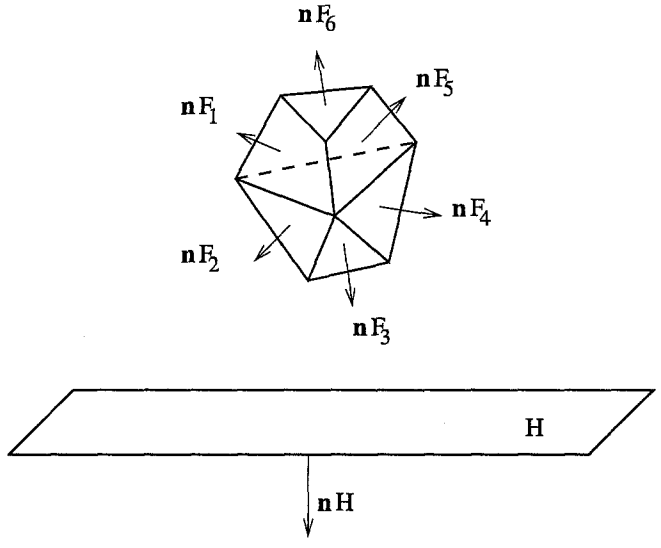


Figure 1. Configuration of a plane H and a polyhedron P.

The extreme vertex problem can be formulated as follows. Let h_1 be a point contained on the plane H. Then, the plane H is given by

$$\langle x - h_1, \mathbf{n}H\rangle = 0 \qquad \text{or} \qquad \langle x, \mathbf{n}H\rangle = \langle h_1, \mathbf{n}H\rangle,$$

where $\langle \cdot, \cdot \rangle$ is the inner product of two vectors. It is well known that for a point $p \in R^3$, $D(p) = \langle p - h_1, \mathbf{n}H\rangle$ is the signed distance of the point p from the plane $\langle x - h_1, \mathbf{n}H\rangle = 0$. The function $D(p)$ has a positive or negative value according as the point p lies in the side of H pointed by $\mathbf{n}H$ or in its opposite side. Hence, $D(v_i) = \langle v_i - h_1, \mathbf{n}H\rangle \leq 0$ for all vertices $v_i \in P$. Since the extreme vertex v_h of P with respect to H has the minimum distance from the plane H, the extreme vertex satisfies the following equation:

$$\begin{aligned}\langle v_h - h_1, \mathbf{n}H\rangle &= \min_{i=1,\ldots,m} \{|\langle v_i - h_1, \mathbf{n}H\rangle|\} \\ &= \max_{i=1,\ldots,m} \{\langle v_i - h_1, \mathbf{n}H\rangle\},\end{aligned}$$

where m is the number of vertices in P. It is equivalent to

$$\langle v_h, \mathbf{n}H\rangle = \max_{i=1,\ldots,m} \{\langle v_i, \mathbf{n}H\rangle\}.$$

Therefore, the extreme vertex problem is to find the maximum of $\langle x, \mathbf{n}H\rangle$ over all vertices in P. This is a well-known linear programming problem [17]: given a convex polyhedron P of n faces, and an objective function $z(x) = \langle x, \mathbf{n}H\rangle$, find the vertex of P that maximizes the function $z(\cdot)$.

In order to efficiently find the extreme vertex, we exploit the point-plane duality. Consider a transformation T that maps a point $p = (p_1, p_2, p_3)$ to a plane $\langle p, x\rangle = p_1x_1 + p_2x_2 + p_3x_3 = 1$, and vice versa [17]. This transformation gives the dual D of the polyhedron P: Every vertex v_h of P corresponds to a face Dv_h of D, and every face F_i of P does to a vertex DF_i of D. Without loss of generality, we may assume that P contains the origin in its interior. Otherwise, we can always translate P and H so that the center of mass of P lies on the origin, since an extreme vertex with respect to H is translation-invariant.

For convenience, we relabel the faces containing $v_h = (x_h, y_h, z_h)$ in P so that they form a cycle $(F_{h,0}, F_{h,1}, \ldots, F_{h,k-1})$, i.e., $F_{h,j}$ and $F_{h,j+1}$ for $0 \le j < k$ share an edge of P, where the second subscripts are taken modulo k. v_h is transformed to a plane

$$\langle v_h, x\rangle = x_hx_1 + y_hx_2 + z_hx_3 = 1.$$

Since $\mathbf{n}F_{h,j}$ is the unit normal vector of $F_{h,j}$ of P, the plane containing $F_{h,j}$ is given by

$$\langle \mathbf{n}F_{h,j}, x\rangle = d_{h,j} \qquad \text{or} \qquad \left\langle \frac{\mathbf{n}F_{h,j}}{d_{h,j}}, x \right\rangle = 1,$$

where $d_{h,j}$ is the distance to the face $F_{h,j}$ from the origin. Since v_h is a vertex of $F_{h,j}$,

$$\left\langle v_h, \frac{\mathbf{n}F_{h,j}}{d_{h,j}} \right\rangle = 1.$$

That is, $DF_{h,j} = \mathbf{n}F_{h,j}/d_{h,j}$ is contained in the plane $\langle v_h, x\rangle = 1$. In fact, $DF_{h,j}$ is a vertex of Dv_h corresponding to $F_{h,j}$ of P [17]. Furthermore,

$$\langle v_h, x\rangle = \|v_h\| \left\langle \frac{v_h}{\|v_h\|}, x \right\rangle = 1 \qquad \text{or} \qquad \left\langle \frac{v_h}{\|v_h\|}, x \right\rangle = \frac{1}{\|v_h\|}.$$

That is, the convex polygon Dv_h corresponding to a vertex v_h of P is contained in the plane $\langle v_h, x\rangle = 1$ whose distance from the origin is $1/\|v_h\|$. In general, the transformation T maps a point p at distance d from the origin to the hyperplane at distance $1/d$ from the origin, that is perpendicular to the ray emanating from the origin and passing through p. It is well known that D is also a convex polyhedron containing the origin in its interior. Moreover, Dv_h is a convex polygon for each vertex v_h of P.

Due to the point-plane duality, $DF_{h,j}$s are the vertices of the convex polygon Dv_h, and the line segment $DE_{h,j}$ joining two vertices, $DF_{h,j}$ and $DF_{h,j+1}$, is an edge of DV_h. As shown in Figure 2, the ray from the origin to $DF_{h,j}$ intersects the Gauss sphere at $\mathbf{n}F_{h,j}$. Therefore, DV_h is projected onto the sphere as a spherical region S_{v_h} whose boundary can be represented by a cycle of spherical points and edges $(\mathbf{n}F_{h,0}, SE_{h,0}, \mathbf{n}F_{h,1}, SE_{h,1}, \ldots, \mathbf{n}F_{h,k-1}, SE_{h,k-1}, \mathbf{n}F_{h,0})$, where each spherical edge $SE_{h,j}$, $1 \le j < k$ is a segment of a great circle joining $\mathbf{n}F_{h,j}$ and $\mathbf{n}F_{h,j+1}$. We call S_{v_h} a spherical extreme vertex region corresponding to v_h. The spherical regions S_{v_h} for all $v_h \in P$ give rise to a partition of the Gauss sphere. This partition is said to be a spherical extreme vertex diagram. Now, we show that the spherical extreme vertex diagram can be constructed in $O(n)$ time.

Let $H_{h,j}$ be the plane containing the origin, $\mathbf{n}F_{h,j}$, and $\mathbf{n}F_{h,j+1}$, $0 \le j < k$. $H_{h,j}$ divides the space into two half-spaces, $H^+_{h,j}$ and $H^-_{h,j}$. Let $H^+_{h,j}$ be the half-space containing Dv_h. The intersection of half-spaces $H^+_{h,j}$, $0 \le j < k$, is a cone with the apex at the origin. Therefore, the spherical region, that is, the projection of Dv_h onto the sphere, is the intersection of the sphere and the cone. The boundary of this region is represented by a sequence of points $(\mathbf{n}F_{h,0}, \mathbf{n}F_{h,1}, \ldots, \mathbf{n}F_{h,k-1})$ such that $\mathbf{n}F_{h,j}$ and $\mathbf{n}F_{h,j+1}$ are joined by a geodesic arc for all $0 \le j < k$. Moreover, the spherical region is a simple spherical polygon on the sphere. Hence, the sequence of the line segment joining $\mathbf{n}F_{h,j}$ and $\mathbf{n}F_{h,j+1}$, $0 \le j < k$, forms a simple closed

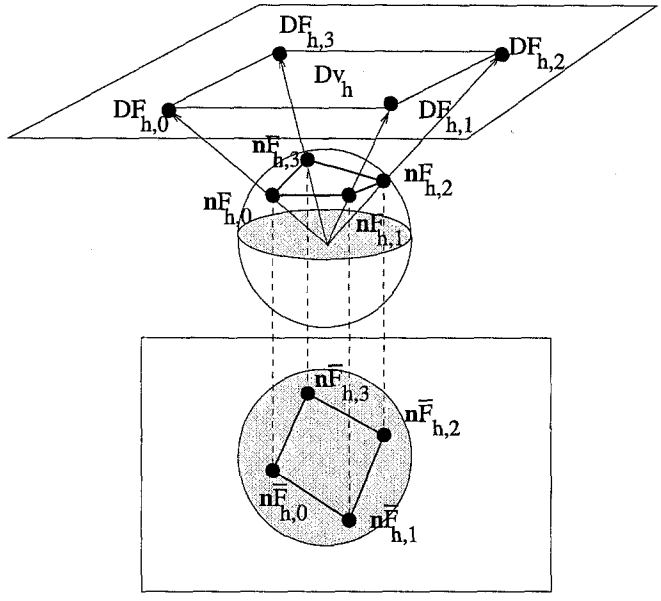


Figure 2. Point-plane duality and projections.

piecewise linear curve. We show that its orthographic projection onto a plane parallel to Dv_h is a convex polygon.

LEMMA 1. *The orthographic projection of the closed piecewise linear curve* $(\mathbf{n}F_{h,0}, \mathbf{n}F_{h,1}, \ldots, \mathbf{n}F_{h,k-1})$ *onto a plane parallel to* Dv_h *is a convex polygon.*

PROOF. Let $(\mathbf{n}\bar{F}_{h,0}, \mathbf{n}\bar{F}_{h,1}, \ldots, \mathbf{n}\bar{F}_{h,k-1})$ be the projection of the curve, where $\mathbf{n}\bar{F}_{h,j}$, $0 \leq j < k$, is the projected image of $\mathbf{n}F_{h,j}$. Suppose that the projection is not convex. Since it is a simple closed curve, there would exist one or more vertices of the projection contained in the interior of the convex hull of the projection. Take any of such vertices, say $\mathbf{n}\bar{F}_{h,j}$ for some $0 \leq j < k$. Then, it must be contained in the interior of the triangle $(\mathbf{n}\bar{F}_{h,a}, \mathbf{n}\bar{F}_{h,b}, \mathbf{n}\bar{F}_{h,c})$, where $0 \leq a < b < c < k$. The inverse projection of the triangle onto the sphere gives a spherical triangle $(\mathbf{n}F_{h,a}, \mathbf{n}F_{h,b}, \mathbf{n}F_{h,c})$, that contains $\mathbf{n}F_{h,j}$ as an interior point. When the spherical triangle is transformed back onto the plane containing Dv_i, $DF_{h,j}$ lies in the interior of the triangle $(DF_{h,a}, DF_{h,b}, DF_{h,c})$, that is completely contained in Dv_i. Thus, $DF_{h,j}$ is an interior point of Dv_i, which contradicts that Dv_i is a convex polygon. Hence, the result holds true. ∎

Aggarwal *et al.* [18] showed that the convex hull of n points can be found in $O(n)$ time if their projections onto a plane are the vertices of a convex polygon. By Lemma 1, the points $\mathbf{n}F_{h,j}$, $0 \leq j < k$, satisfy this condition. Therefore, their convex hull can be constructed in $O(k)$ time. The piecewise linear curve has another nice property that is very useful to construct the convex hull of $\mathbf{n}F_i$s in $O(n)$ time.

LEMMA 2. *The closed piecewise linear curve* $(\mathbf{n}F_{h,0}, \mathbf{n}F_{h,1}, \ldots, \mathbf{n}F_{h,k-1})$ *consists of a subset of edges of the convex hull of* $\mathbf{n}F_i$ *for all* $i = 1, 2, \ldots, n$.

PROOF. Since $\mathbf{n}F_i$s lie on the Gauss sphere, all of them are extreme points, i.e., the vertices of the convex hull of $\mathbf{n}F_i$s. Since $\mathbf{n}F_{h,j} \in \{\mathbf{n}F_1, \mathbf{n}F_2, \ldots, \mathbf{n}F_f\}$ for all $0 \leq j < k$, $\mathbf{n}F_{h,j}$s are also extreme points. We will be done if we show that the line segment joining $\mathbf{n}F_{h,j}$ and $\mathbf{n}F_{h,j+1}$ is an edge of the convex hull of $\mathbf{n}F_i$s.

Remember that the projection of each face Dv_h of the dual D of the convex polyhedron P is a spherical region $(\mathbf{n}F_{h,0}, \mathbf{n}F_{h,1}, \ldots, \mathbf{n}F_{h,k-1})$. This region is the intersection of the Gauss sphere and the cone bounded by the planes $H_{h,j}$ for all $0 \leq j < k$. The geodesic arc connecting $\mathbf{n}F_{h,j}$ and $\mathbf{n}F_{h,j+1}$ lies on $H_{h,j}$, and so does the line segment joining them. For every $v_h \in P$, the

cone CO_h is well defined, i.e., $CO_h = \bigcap_j H^+_{h,j}$. The set of all these cones partitions the sphere into n disjoint spherical regions. None of the spherical regions contain a spherical image $\mathbf{n}F_i$, $1 \leq i \leq n$ in its interior.

Suppose, for a contradiction, that a line segment of the curve is not an edge of the convex hull of $\mathbf{n}F_i$s, say the line segment joining $\mathbf{n}F_{h,l}$ and $\mathbf{n}F_{h,l+1}$ for some $0 \leq l < k$. Then, it must be a diagonal. Therefore, the line segment excepting its end points $\mathbf{n}F_{h,l}$ and $\mathbf{n}F_{h,l+1}$ is completely contained in the interior of the cone CO_h. If we project the line segment back to Dv_h, it becomes a diagonal of Dv_h, which is a contradiction since the inverse projection of the line segment is an edge of the convex polyhedron Dv_h. Hence, the result follows immediately. ∎

Now, we are ready to describe how to construct the convex hull of the Gauss images $\mathbf{n}F_i$s of the normal vectors of the faces of P in $O(n)$ time. By Lemma 2, the set of all such curves, that result from the faces of D, partitions the boundary of the convex hull into n disjoint regions. Lemma 1 guarantees that the convex hull of each of these regions can be found in linear time. Therefore, the convex hull of $\mathbf{n}F_i$s can be constructed in $O(n)$ time once all such curves are identified. The curves are obtained in $O(n)$ time by simply projecting the faces of D onto the Gauss sphere with the origin as the projection center. Given the convex hull of $\mathbf{n}F_i$s, the spherical extreme vertex diagram can be computed in $O(n)$ because it has the same topology as the convex hull.

THEOREM 1. *The convex hull of $\mathbf{n}F_i$s, $1 \leq i \leq n$, can be found in $O(n)$ time. Moreover, their spherical extreme vertex diagram can be constructed in the same time bound.*

Using this diagram, we can easily transform the linear programming problem to the ray shooting problem [17]: given a ray R emanating from the origin in the direction given by $\mathbf{n}H$, find the dual face Dv_h of D that is intersected by this ray. Let p_i, $0 \leq i \leq m$ be the intersection point of the line L containing the ray R with the hyper plane, $H_i : \langle v_i, x\rangle = 1$ containing a dual face Dv_i, respectively, and d_i be the signed distance of p_i from the origin. Then, the point can be represented by $p_i = d_i\mathbf{n}H$. By the inverse transformation, each point p_i is transformed to a hyper plane, $T^{-1}(p_i) : \langle p_i, x\rangle = 1$, perpendicular to R whose distance from the origin is $1/d_i$. That is,

$$\langle p_i, x\rangle = 1 \qquad \text{or} \qquad \langle \mathbf{n}H, x\rangle = \frac{1}{d_i}.$$

Since p_i is contained in H_i, it satisfies the plane equation

$$\langle v_i, p_i\rangle = 1 \qquad \text{or} \qquad \langle v_i, \mathbf{n}H\rangle = \frac{1}{d_i}. \tag{1}$$

Hence, each vertex v_i of P is contained in the hyper plane $T^{-1}(p_i) : \langle \mathbf{n}H, x\rangle = 1/d_i$. Suppose that the ray R intersects a dual face Dv_h at p_h. It is equivalent to that the ray R first intersects the hyper plane H_h containing the face D_{v_h}, since the dual D is the intersection of half-spaces containing the origin. Hence, p_h is the closest point to the origin among all p_is on the ray R, i.e., $d_h \leq d_i$. For all p_is on the opposite ray, $d_h > d_i$ since d_i is negative. From equation (1), we have the following inequalities:

$$\langle v_h, \mathbf{n}H\rangle = \frac{1}{d_h} \geq \frac{1}{d_i} = \langle v_i, \mathbf{n}H\rangle, \qquad \text{for all } i.$$

Therefore, we have the following equality: $\langle v_h, \mathbf{n}H\rangle = \max_i\{\langle v_i, \mathbf{n}H\rangle\}$. That is, the objective function $z(x) = \langle x, \mathbf{n}H\rangle$ has the maximum at v_h. Therefore, the linear programming problem is equivalent to the ray shooting problem.

In order to efficiently find the dual face Dv_h, we utilize the spherical extreme vertex diagram, that is, the projection of D onto the Gauss sphere [19]. If the ray R passing through $\mathbf{n}H$ intersects a dual face Dv_h, then R also intersects the spherical region S_{v_h} corresponding to Dv_h at $\mathbf{n}H$. That is, $\mathbf{n}H$ is contained in the region S_{v_h}. Hence, the ray shooting problem can be transformed

to a spherical point location problem for the spherical diagram. The point location problem can be solved in $O(\log n)$ time [19]. If $\mathbf{n}H$ coincides with a vertex $\mathbf{n}F_{h,j}$ of the spherical extreme vertex diagram, then the plane H is parallel to the face $F_{h,j}$ of P. That is, all of the vertices of $F_{h,j}$ are the extreme vertices of P with respect to H. If $\mathbf{n}H$ lies on the common great circle arc of two spherical extreme vertex regions Sv_h and Sv_j, then both vertices v_h and v_j corresponding to the regions are the extreme vertices of P. Moreover, if $\mathbf{n}H$ is contained in the interior of a spherical extreme vertex region Sv_h, then its corresponding vertex v_h is the extreme vertex of P with respect to H. Hence, we can find an extreme vertex of P with respect to H by solving the spherical point location problem after constructing the spherical extreme vertex diagram.

THEOREM 2. *The extreme vertex of P with respect to H can be computed in $O(\log n)$ time after $O(n)$ time and space preprocessing.*

3. QUATERNION AND SPHERICAL CURVES

The spherical extreme vertex diagram plays a central role in determining the extreme vertex of a polyhedron undergoing a rigid motion with respect to a fixed plane H. The rigid motion of P consists of translation and rotation. Since H is a fixed infinite plane, the extreme vertex of P with respect to H is translation-invariant. Therefore, it is sufficient to consider only rotation to determine the extreme vertices.

We use the unit quaternion representation to describe rotation of P [20–22]. A unit quaternion $\mathbf{Q} \in R^4$ consists of a scalar part and a 3D-vector. We denote the scalar part of quaternion $\mathbf{Q}$ by q_0 and the vector part by $\mathbf{q}$, and to simplify notations, the scalar and vector part will be combined using an ordered pair, so that $\mathbf{Q} = (q_0, \mathbf{q})$. It is well known that there is a correspondence between unit quaternions and rotations: suppose that we rotate a vector $\mathbf{v}$ about an axis $\mathbf{r}$ by an angle of size $\|\mathbf{r}\|$, thereby forming a new vector $\mathbf{v}'$. Then, the rotated vector $\mathbf{v}'$ is given by

$$\mathbf{v}' = \mathbf{Q}\,\mathbf{v}\,\mathbf{Q}^{-1},$$

where

$$\mathbf{Q} = \left(\cos\frac{\|\mathbf{r}\|}{2}, \frac{\mathbf{r}}{\|\mathbf{r}\|}\sin\frac{\|\mathbf{r}\|}{2}\right).$$

Let $\mathbf{Q}(t)$ be a unit quaternion curve that gives the orientation of P at time t [23]. While the infinite plane H is fixed, the polyhedron P rotates about $\mathbf{r}(t)$ by $\|\mathbf{r}(t)\|$ (i.e., $\mathbf{Q}(t)$). Every extreme vertex of P with respect to H is dependent on the relative orientation of P to H. Therefore, we can fix P and rotate H about $-\mathbf{r}(t)$ by $\|\mathbf{r}(t)\|$ (i.e., $\mathbf{Q}^{-1}(t)$). Without loss of generality, we assume that H moves around P, hereafter. Under this assumption, $\mathbf{n}H$ moves and $\mathbf{n}F_k$s are fixed. The rotation of H for $0 \le t \le 1$ gives an orientation path $\mathbf{n}H(t)$ on the Gauss sphere.

4. GENERATION OF DISTANCE FUNCTION

The spherical path $\mathbf{n}H(t)$ crosses over a sequence of the spherical extreme vertex regions as the polyhedron P rotates according to a unit quaternion curve $\mathbf{Q}(t)$. While the path is intersecting a spherical region S_i, the vertex v_i corresponding to the region is the extreme vertex of P with respect to H. Hence, the distance of v_i to H is itself that of the polyhedron P. Therefore, if we keep track of the sequence of time instances at which the path enters or leaves a region, then we are able to construct a distance function of time between P and H.

In general, the sequence may be found by computing the intersection points of the path and the edges of spherical regions. However, since most of the intersection tests utilize numerical methods, it takes much time to accomplish the intersection test. This is more vital in animations for a simulation which requires real time computation. Hence, in order to overcome such a

problem, we propose an alternative which is useful in animations. It is to determine whether the segment of $\mathbf{n}H(t)$ for a given time interval $[t_s, t_e]$ is contained in only one region or two adjacent regions. We call the time interval an interval of same-region type or adjacent-region type according to whether its corresponding segment is contained in one region or two adjacent regions, respectively. If the time interval is neither of the same-region type nor of the adjacent-region type, then it is said to be of an undetermined type. We show that the classification is accomplished by comparing the arc-length of the path $\mathbf{n}H(t)$ with the spherical distance defining a spherical ellipse.

4.1. Determination of Segment Types

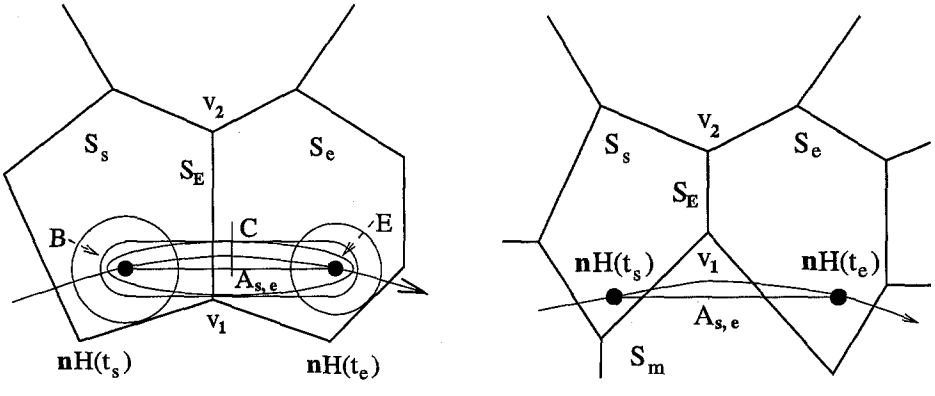
Let t_s and t_e be the start and the end time instances of a given time interval $I_{s,e} = [t_s, t_e]$. Let S_s and S_e be the spherical extreme vertex regions to which $\mathbf{n}H(t_s)$ and $\mathbf{n}H(t_e)$ belong, respectively. If S_s is neither equal nor adjacent to S_e, then the segment of the path during the time interval $I_{s,e}$ must pass over regions other than S_s and S_e. For this case, we subdivide the time interval $I_{s,e}$ into two subintervals and then apply the same determination process to both of them. Otherwise, there is a possibility for the segment to be contained in the union of S_s and S_e. We first investigate the possibility for the adjacent-region case: S_s is adjacent to S_e.

ADJACENT-REGION CASE. Suppose that S_s is adjacent to S_e with the common spherical edge $S_E = (v_1, v_2)$. Consider a great circular arc $A_{s,e}$ joining $\mathbf{n}H(t_s)$ and $\mathbf{n}H(t_e)$. If the arc $A_{s,e}$ intersects the common spherical edge S_E as shown in Figure 3a, then we may construct a spherical ellipse contained in the union of S_s and S_e. First of all, consider a rounded bounding box B of the great circular arcs such that its horizontal lines are circular arcs parallel to $A_{s,e}$ and both its rounded parts are half-circles of radius R in geodesic sense, where $R = \min\{r_s, r_e, r_{v_1}, r_{v_2}\}$,

$$
\begin{aligned}
r_s &= \min_{x \in Bd(S_s)} \{G(\mathbf{n}H(t_s), x)\}, \\
r_e &= \min_{x \in Bd(S_e)} \{G(\mathbf{n}H(t_e), x)\}, \\
r_{v_1} &= G\left(S_{v_1}, A_{s,e}\right), \\
r_{v_2} &= G\left(S_{v_2}, A_{s,e}\right).
\end{aligned}
$$

Here, $G(x, y)$ is the geodesic of x and y on the Gauss sphere and is given by $G(x, y) = \cos^{-1}\langle x, y\rangle$. Let C be an intersection point of the boundary of B and the perpendicular bisector of $A_{s,e}$. Now, we may define a spherical ellipse E as follows:

$$
E = \left\{x \in S^2 \mid G(\mathbf{n}H(t_s), x) + G(\mathbf{n}H(t_e), x) = GS_C\right\},
$$



(a) The base edge $A_{s,e}$ intersects the common edge S_E.

(b) $A_{s,e}$ does not intersect S_E.

Figure 3. Adjacent-region cases.

where GS_C is the sum of spherical distances to C of $\mathbf{n}H(t_s)$ and $\mathbf{n}H(t_e)$. We can easily show that the ellipse E is contained in the bounding box B.

Due to the comparison of arc-length of $\mathbf{n}H(t)$ with GS_C, we are now ready to show whether the time interval $I_{s,e}$ is of a adjacent-region type. If $M(t_e - t_s) \leq GS_C$, where M is the upper bound of speed of $\mathbf{n}H(t)$ (see Appendix A for a detailed derivation), then the arc-length of $\mathbf{n}H(t)$ for the time interval $I_{s,e}$ is equal to or less than GS_C;

$$A_{\mathbf{n}H(t)}(t_s, t_e) = \int_{t_s}^{t_e} \|\mathbf{n}H'(t)\| \, dt \leq M(t_e - t_s) \leq GS_C.$$

That is, the segment of $\mathbf{n}H(t)$ is contained in the ellipse E. Hence, the segment is also contained in the union of S_s and S_e. Therefore, this time interval $I_{s,e}$ is classified to be of an adjacent-region type. Otherwise, it is hard to determine whether the segment is contained in the union. For this case, $I_{s,e}$ is classified to be of an undetermined type. Hence, we need to subdivide the time interval $I_{s,e}$ succeedingly.

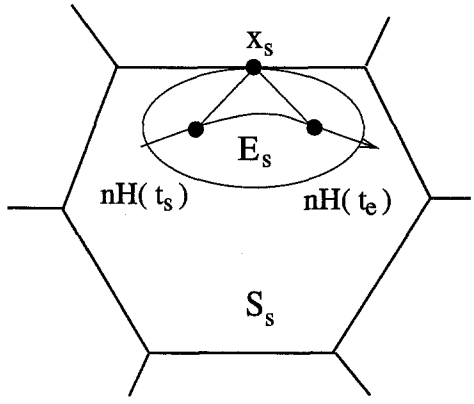


Figure 4. Same-region type.

SAME-REGION CASE. Suppose that S_s coincides with S_e. Let $Bd(S_s)$ be the boundary of S_s represented by a cycle of spherical edges $(S_{e_1}, S_{e_2}, \ldots, S_{e_{m_s}})$, where m_s is the number of spherical edges of S_s. In order to define a spherical ellipse contained in S_s, we identify a point x_s on $BD(S_s)$ which has the smallest sum of spherical distances to $\mathbf{n}H(t_s)$ and $\mathbf{n}H(t_e)$,

$$G(\mathbf{n}H(t_s), x_s) + G(\mathbf{n}H(t_e), x_s) = \min_{x \in Bd(S_s)} \{G(\mathbf{n}H(t_s), x) + G(\mathbf{n}H(t_e), x)\}.$$

Here, we denote the sum by $G_s = G(\mathbf{n}H(t_s), x_s) + G(\mathbf{n}H(t_e), x_s)$. Since every spherical edge of a region is a geodesic (i.e., a great circular arc) and every spherical extreme vertex region is convex, the point x_s can be computed in $O(m_s)$ time. Now, we define a spherical ellipse E_s by

$$E_s = \left\{ x \in S^2 \mid G(\mathbf{n}H(t_s), x) + G(\mathbf{n}H(t_e), x) = G_s \right\}.$$

Then, the ellipse is contained in the region S_s by the definition of G_s and the convexity of S_s and E_s.

Now, in order to check whether the time interval $I_{s,e}$ is of a same-region type, we compare the arc-length of $\mathbf{n}H(t)$ for $I_{s,e}$ with G_s. If the interval satisfies the inequality $M(t_e - t_s) \leq G_s$, where M is the upper bound of speed of $\mathbf{n}H(t)$ for I_{se} (see Appendix A for a detailed derivation), then the arc-length of $\mathbf{n}H(t)$ for I_{se} is also equal to or less than G_s,

$$A_{\mathbf{n}H(t)}(t_s, t_e) = \int_{t_s}^{t_e} \|\mathbf{n}H'(t)\| \, dt \leq M(t_e - t_s) \leq G_s.$$

Thus, the segment of $\mathbf{n}H(t)$ must be contained in the spherical ellipse E_s. Hence, this time interval I_{se} is classified to be of a same-region type. Otherwise, at the present time, it is hard to determine whether the segment is contained in S_s only. Hence, $I_{s,e}$ is classified to be of an undetermined type.

4.2. Distance Function

Now, we are ready to obtain the distance function $D(t)$ between a convex polyhedron P and a fixed plane H during a time interval of same-region and adjacent-regions types. The segment of $\mathbf{n}H(t)$ during a time interval of same-region type is contained in only one region S_i, whereas the segment during a time interval of adjacent-regions type is contained in the union of two adjacent regions S_j and S_{j+1}. The vertex v_i of P assigned to S_i is the extreme vertex of P with respect to H, whereas one of two vertices v_j and v_{j+1} is the extreme vertex during the time interval. Hence, in order to construct the distance function $D(t)$, it is sufficient to investigate the movement of the extreme vertices corresponding to the regions.

Without loss of generality, we may assume that the given infinite plane H is the xy-plane in R^3 and that each vertex v_i of P is represented by the relative position $\mathbf{r}_i$ about the centroid c_p of P as follows:

$$\mathbf{r}_i = (r_{i1}, r_{i2}, r_{i3}) = v_i - c_p.$$

Undergoing the rigid motion of P, the position $p_i(t)$ of v_i can be represented by

$$p_i(t) = T(t) + \mathbf{Q}(t)\,\mathbf{r}_i\,\mathbf{Q}^{-1}(t),$$

where $T(t) = (T_1(t), T_2(t), T_3(t))$ is the translation of P. Hence, the distance function $d_i(t)$ between v_i and H is the z-component of $p_i(t)$ and represented by

$$\begin{aligned} d_i(t) = T_3(t) &+ 2r_{i1}\left(-q_0(t)q_2(t) + q_1(t)q_3(t)\right) + 2r_{i2}\left(q_0(t)q_1(t) + q_2(t)q_3(t)\right) \\ &+ r_{i3}\left(q_0^2(t) - q_1^2(t) - q_2^2(t) + q_3^2(t)\right). \end{aligned}$$

Therefore, the distance function $D(t)$ between P and H during a time interval $[t_s, t_e]$ according to its type can be represented by (see Appendix B)

$$D(t) = \begin{cases} d_i(t), & \text{of same-region type,} \\ \min\left\{d_j(t), d_{j+1}(t)\right\}, & \text{of adjacent-regions type.} \end{cases}$$

5. COLLISION DETECTION OF A CONTINUOUS TYPE

In this section, we present an efficient collision detection algorithm based on the determination of time intervals and interval Newton method. The algorithm is accomplished by passing through the following three steps: subdivision of time intervals, interval Newton method [16], and display of scenes. As the first step, we subdivide the given time interval $[0, 1]$ recursively until we obtain a time interval $[0, t_{s_1}]$ of same-region type. We first determine the type of the given interval $[0, 1]$. If it is of same-region type, then the subdivision process is done. Otherwise, we subdivide $[0, 1]$ to two half-intervals and save the second half-interval on a stack. And then, we determine the type of the first half-interval. If the type is not of same-region, then we repeat the above subdivision and type-determination until a time interval $[0, t_{s_1}]$ of same-region type is obtained.

The segment of $\mathbf{n}H(t)$ during the time interval $[0, t_{s_1}]$ is obtained in the spherical region S_0 including the initial spherical point $\mathbf{n}H(0)$, and thus, the distance function $d_0(t)$ is generated by the vertex v_0 corresponding to S_0. As the second step, we apply the interval Newton method to the distance function in order to check whether collisions occur. The collision time is the first root of equation $d_0(t) = 0$. The interval Newton method not only reports the existence of roots in a few of the iterations, but also computes the first root without sorting the roots if they exist. Hence, it is a method appropriate for collision detection problem.

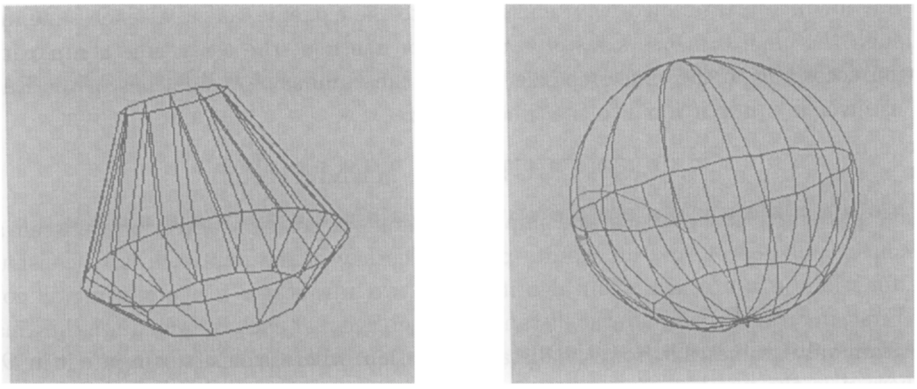
If the interval $[0, t_{s_1}]$ has no collision times, then we only display the moving polyhedron without computing the distance at each step until the end time instance t_{s_1}. When simulation time reaches t_{s_1}, we take the top element $[t_{s_1}, t_{s_2}]$ out of the stack. If this new interval is of same-region or adjacent-regions type, then we continue the simulation at the second step: applying

the interval Newton method to this time interval. Otherwise, we repeat the simulation beginning with the first step: subdivision process for this time interval.

If the time interval $[0, t_{s_1}]$ has a collision time t_c, then we only display the moving polyhedron without computing the distance until the collision time t_c. When it comes to the collision time, we compute the impulsive force and torque with the linear and angular velocities of the polyhedron P, and thus, obtain new translation and rotation of P. That is, a new spherical path $\mathbf{n}H(t)$ is generated on the spherical extreme vertex diagram. With the new path, we repeat the whole process beginning with the first step: subdivision process for the remaining time interval $[t_c, 1]$.

6. EXPERIMENTAL RESULTS

We first simulate a throwing pot and compute the time complexity of the simulation during a time interval $[0.0, 3.0]$. Figure 5a shows a throwing pot with the linear velocity $v = (10.0, 5.0, 15.0)$ and with the angular velocity $\omega = (-1.789, 1.123, -0.789)$. The spherical extreme vertex diagram of the pot is represented by a wire-frame, and the spherical path $\mathbf{n}H(t)$ generated by its rotation is drawn on the spherical diagram as shown in Figure 5b. We choose the upper bound M of the speed of $\mathbf{n}H(t)$ as $M = \sqrt{3}\|\omega\| = 3.905446$ and the threshold ϵ of the interval Newton method as $\epsilon = 0.000001$. The throwing pot first collides with the floor at time $t_{\alpha_1} = 1.647786$, and then bounces off with a new linear velocity $v = (10.0, 5.0, 12.28)$ and with a new angular velocity $\omega = (1.12, 1.31, -0.21)$. And their second collision time is $t_{\alpha_2} = 2.918145$, where the pot bounces off with a new linear velocity $v = (10.0, 5.0, 5.93)$ and with a new angular velocity $\omega = (-0.89, -6.44, -0.49)$.



(a) A throwing pot. (b) Its spherical extreme vertex diagram.

Figure 5.

The experimental results of our algorithm with seven polyhedra are shown in Tables 1 and 2. Table 1 shows that the collision times of our algorithm are independent of the time step size, whereas those of the incremental algorithms in [9] are not. In our algorithm, the time intervals applied to the numerical method are decided according to the spherical extreme regions which the spherical curve passes through, so the sequence of the time intervals is independent of the time step size of animation. Hence, the collision time is invariant under the change of the time step size of animation. On the other hand, in incremental algorithms, each time step of animation itself is applied to the numerical method, so the collision time may be changed according to the time step size due to the numerical error.

Table 2 shows the time complexities of collision detection algorithms without rendering. The time complexity of our algorithm is independent of the time step size, whereas that of the incremental algorithm [9] in case of $\Delta t = 0.001$ is five times as long as that in $\Delta t = 0.005$. Our algorithm does not need to compute the extreme vertex at a time instance because we already

Table 1. Collision times between the pot and the floor during [0.0, 3.0].

Time Step Size	$\Delta t = 0.005$		$\Delta t = 0.001$	
Collision Times	t_{α_1}	t_{α_2}	t_{α_1}	t_{α_2}
Our Algorithm	1.647786	2.918145	1.647786	2.918145
Incremental Algorithm	1.647788	2.918164	1.647781	2.917781

Table 2. The computation time without rendering.

	Our Algorithm		Incremental Algorithm	
Polygonal Type (No. of Faces)	$\Delta t = 0.005$ (Seconds)	$\Delta t = 0.001$ (Seconds)	$\Delta t = 0.005$ (Seconds)	$\Delta t = 0.001$ (Seconds)
Cube (6)	0.159435	0.158708	2.496181	12.479868
Octahedron (8)	0.156491	0.162084	2.491545	12.507955
Con (13)	0.307315	0.307915	2.612410	12.800396
Pot (50)	0.411429	0.423491	5.450509	27.569424
Sphere (64)	1.299005	1.292649	6.478204	31.792561
Sphere (100)	1.482894	1.497369	11.416183	50.046286
Sphere (200)	4.379463	4.377843	17.176759	80.620888

know them in the precess of distance generation, so the Euclidean distance can be computed in a constant time only by evaluating the value from the distance function. That is, the computation of the Euclidean distance is irrelevant to the time step size of animation. Hence, the change of time step size does not affect the time complexity of our collision detection algorithm. On the other hand, the incremental algorithm must compute the extreme vertex at each time instance. Even though the algorithm has a faster computation method, the total time complexity must be changed according to the time step size because the number of evaluations of the Euclidean distance is inversely proportional to the time step size.

7. CONCLUDING REMARKS

We have presented an approach and implementation for collision detection between a moving 3D convex polyhedron and an infinite plane. The key of the approach is to construct the distance function between them. Our algorithm has two advantages for computing all collisions exactly. First, in order to compute the distance between them at each intermediate time instance, our algorithm only evaluates it from the distance function, whereas others compute it in $O(\log n)$ time, where n is the number of vertices of the polyhedron. Hence, the total time complexity of the collision detection algorithm is independent of the time step size, so this method may be useful to the environment in which an object collides with another object several times. Second, our algorithm applies the interval Newton method to wider intervals than the animation time step, which are irrelevant to time step size of animation. The time intervals are decided according to the spherical extreme regions which the spherical curve passes through, so the sequence of the time intervals is independent of the time step size of animation. Hence, the collision times are invariant under the change of the time step size and can be computed within a given tolerance ϵ.

APPENDIX A
SPHERICAL CURVE AND ITS VELOCITY

The spherical path $\mathbf{n}H(t)$ is represented by

$$\begin{aligned}\mathbf{n}H(t) &= \mathbf{Q}^{-1}(t)\,\mathbf{n}H\,\mathbf{Q}(t)\\ &= \begin{bmatrix} 2\,(q_0(t)q_2(t) - q_1(t)q_3(t)) \\ -2\,(q_0(t)q_1(t) + q_2(t)q_3(t)) \\ -q_0^2(t) + q_1^2(t) + q_2^2(t) - q_3^2(t) \end{bmatrix}^{\top},\end{aligned}$$

where $\mathbf{Q}(t) = (q_0(t), q_1(t), q_2(t), q_3(t))$ and $\mathbf{Q}^{-1}(t) = (q_0(t), -q_1(t), -q_2(t), -q_3(t))$. The arc-length of $\mathbf{n}H(t)$ between t_s and t_e, $A_{\mathbf{n}H}(t_s, t_e)$, is defined as follows:

$$A_{\mathbf{n}H}(t_s, t_e) = \int_{t_s}^{t_e} \|\mathbf{n}H'(t)\| \, dt,$$

where $\|\mathbf{n}H'(t)\|$ is the speed of $\mathbf{n}H(t)$. Since the speed has a square root term, it is hard to compute the arc-length analytically. Hence, we need to estimate an upper bound of $\|\mathbf{n}H'(t)\|$.

The tangent vector of $\mathbf{n}H(t)$ at time t is obtained by component-wise differentiation

$$\mathbf{n}H'(t) = 2 \begin{bmatrix} q_0'(t)q_2(t) + q_0(t)q_2'(t) - q_1'(t)q_3(t) - q_1(t)q_3'(t) \\ -q_0'(t)q_1(t) - q_0(t)q_1'(t) - q_2'(t)q_3(t) - q_2(t)q_3'(t) \\ -q_0(t)q_0'(t) + q_1(t)q_1'(t) + q_2(t)q_2'(t) - q_3(t)q_3'(t) \end{bmatrix}^{\top}.$$

Then,

$$\begin{aligned}
\frac{\|\mathbf{n}H'(t)\|^2}{4} &= (q_0'(t))^2 \left(q_0^2(t) + q_1^2(t) + q_2^2(t)\right) + (q_1'(t))^2 \left(q_0^2(t) + q_1^2(t) + q_3^2(t)\right) \\
&\quad + (q_2'(t))^2 \left(q_0^2(t) + q_2^2(t) + q_3^2(t)\right) + (q_3'(t))^2 \left(q_1^2(t) + q_2^2(t) + q_3^2(t)\right) \\
&\quad + 2\, q_0'(t)q_3'(t)q_0(t)q_3(t) - 2\, q_0'(t)q_1'(t)q_2(t)q_3(t) \\
&\quad + 2\, q_0'(t)q_2'(t)q_1(t)q_3(t) - 2\, q_2'(t)q_3'(t)q_0(t)q_1(t) \\
&\quad + 2\, q_1'(t)q_3'(t)q_0(t)q_2(t) + 2q_1'(t)q_2'(t)q_1(t)q_2(t) \\
&\leq (q_0'(t))^2 \left(q_0^2(t) + q_1^2(t) + q_2^2(t)\right) + (q_1'(t))^2 \left(q_0^2(t) + q_1^2(t) + q_3^2(t)\right) \\
&\quad + (q_2'(t))^2 \left(q_0^2(t) + q_2^2(t) + q_3^2(t)\right) + (q_3'(t))^2 \left(q_1^2(t) + q_2^2(t) + q_3^2(t)\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_0'(t)|^2 + |q_3'(t)|^2\right) \cdot \left(|q_0(t)|^2 + |q_3(t)|^2\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_0'(t)|^2 + |q_1'(t)|^2\right) \cdot \left(|q_2(t)|^2 + |q_3(t)|^2\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_0'(t)|^2 + |q_2'(t)|^2\right) \cdot \left(|q_1(t)|^2 + |q_3(t)|^2\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_2'(t)|^2 + |q_3'(t)|^2\right) \cdot \left(|q_0(t)|^2 + |q_1(t)|^2\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_1'(t)|^2 + |q_3'(t)|^2\right) \cdot \left(|q_0(t)|^2 + |q_2(t)|^2\right) \\
&\quad + \frac{1}{2} \cdot \left(|q_1'(t)|^2 + |q_2'(t)|^2\right) \cdot \left(|q_1(t)|^2 + |q_2(t)|^2\right) \\
&= \frac{3}{2} \left((q_0'(t))^2 + (q_1'(t))^2 + (q_2'(t))^2 + (q_3'(t))^2\right) \cdot \left(q_0^2(t) + q_1^2(t) + q_2^2(t) + q_3^2(t)\right) \\
&= \frac{3}{2} \|\mathbf{Q}'(t)\|^2 \cdot \|\mathbf{Q}(t)\|^2 \\
&= \frac{3}{2} \|\mathbf{Q}'(t)\|^2.
\end{aligned}$$

Therefore,

$$\|\mathbf{n}H'(t)\| \leq \sqrt{6}\, \|\mathbf{Q}'(t)\|. \tag{2}$$

Now, we estimate the speed of $\mathbf{Q}(t)$. The velocity $\mathbf{Q}'(t)$ satisfies the following equation (see [21,22]):

$$\mathbf{Q}'(t) = \frac{1}{2}\mathbf{Q}(t)\mathbf{w}(t),$$

where $\mathbf{w}(t)$ is the angular velocity of $\mathbf{Q}(t)$. By quaternion multiplication, we have

$$\mathbf{Q}'(t) = \frac{1}{2}\left(-\langle \mathbf{q}(t), \mathbf{w}(t)\rangle,\ q_0(t)\mathbf{w}(t) + \mathbf{q}(t) \times \mathbf{w}(t)\right),$$

where $\langle \mathbf{q}(t), \mathbf{w}(t) \rangle$ and $\mathbf{q}(t) \times \mathbf{w}(t)$ are the inner and cross product of two vectors $\mathbf{q}(t)$ and $\mathbf{w}(t)$, respectively. Then,

$$\|\mathbf{Q}'(t)\|^2 = \frac{1}{4} \langle \mathbf{q}(t), \mathbf{w}(t) \rangle^2 + \frac{1}{4} \|q_0 \mathbf{w}(t) + \mathbf{q}(t) \times \mathbf{w}(t)\|^2. \tag{3}$$

The second term of the right-hand side of (3) is as follows:

$$\begin{aligned} \|(q_0 \mathbf{w}(t),\, \mathbf{q}(t) \times \mathbf{w}(t))\|^2 &= \langle (q_0(t)\mathbf{w}(t),\, \mathbf{q}(t) \times \mathbf{w}(t)),\, (q_0(t)\mathbf{w}(t),\, \mathbf{q}(t) \times \mathbf{w}(t)) \rangle \\ &= \|q_0(t)\mathbf{w}(t)\|^2 + 2\, q_0(t) \langle \mathbf{w}(t), \mathbf{q}(t) \times \mathbf{w}(t) \rangle + \|\mathbf{q}(t) \times \mathbf{w}(t)\|^2 \\ &= \|q_0(t)\mathbf{w}(t)\|^2 + \|\mathbf{q}(t) \times \mathbf{w}(t)\|^2, \end{aligned} \tag{4}$$

since $\mathbf{w}(t)$ and $\mathbf{q}(t) \times \mathbf{w}(t)$ are orthogonal. Plugging equation (4) into equation (3),

$$\begin{aligned} 4\, \|\mathbf{Q}'(t)\|^2 &= \langle \mathbf{q}(t), \mathbf{w}(t) \rangle^2 + \left\| q_0(t)\mathbf{w}(t)^2 \right\|^2 + \left\| \mathbf{q}(t) \times \mathbf{w}(t)^2 \right\|^2 \\ &\leq \|\mathbf{q}(t)\|^2 \|\mathbf{w}(t)\|^2 + q_0^2(t) \|\mathbf{w}(t)\|^2 + \|\mathbf{q}(t)\|^2 \|\mathbf{w}(t)\|^2 \\ &= \|\mathbf{q}(t)\|^2 \|\mathbf{w}(t)\|^2 + \left(q_0^2(t) + \|\mathbf{q}(t)\|^2 \right) \|\mathbf{w}(t)\|^2 \\ &\leq 2 \|\mathbf{Q}(t)\|^2 \|\mathbf{w}(t)\|^2 \\ &\leq 2 \|\mathbf{w}(t)\|^2. \end{aligned}$$

Hence, we have

$$\|\mathbf{Q}'(t)\| \leq \frac{1}{\sqrt{2}} \|\mathbf{w}(t)\|. \tag{5}$$

From equations (2) and (5), we have the following upper bound of speed of $\mathbf{n}H(t)$:

$$\|\mathbf{n}H'(t)\| \leq \sqrt{3} \|\mathbf{w}(t)\|. \tag{6}$$

We denote the upper bound by M.

APPENDIX B
DISTANCE FUNCTION

Let $\mathbf{R}(t) = \mathbf{Q}(t)\mathbf{V}\mathbf{Q}^{-1}(t)$, where

$$\begin{aligned} \mathbf{Q}(t) &= (q_0(t), \mathbf{q}(t)), \\ \mathbf{V} &= (0, \mathbf{v}), \\ \mathbf{Q}^{-1}(t) &= (q_0(t), -\mathbf{q}(t)). \end{aligned}$$

It is well known that the multiplication of two quaternion $\mathbf{P}$ and $\mathbf{Q}$ is given by

$$\mathbf{PQ} = \left(p_0 q_0 - \mathbf{p}\langle \mathbf{p},\, \mathbf{q} \rangle,\, p_0\, \mathbf{q} + q_0\, \mathbf{p} + \mathbf{p} \times \mathbf{q} \right).$$

Hence, we have the following equation: $\mathbf{R} = (r_0, \mathbf{r})$, where

$$\begin{aligned} r_0 &= \langle \mathbf{q} \times \mathbf{v},\, \mathbf{q} \rangle = 0, \\ \mathbf{r} &= \langle \mathbf{q}, \mathbf{v} \rangle + q_0^2 \mathbf{v} + 2 q_0 (\mathbf{q} \times \mathbf{v}) - (\mathbf{q} \times \mathbf{v}) \times \mathbf{q}. \end{aligned} \tag{7}$$

Here,

$$\langle \mathbf{q}, \mathbf{v} \rangle = q_1 v_1 + q_2 v_2 + q_3 v_3, \tag{8}$$

$$\mathbf{q} \times \mathbf{v} = \begin{bmatrix} q_2 v_3 - q_3 v_2 \\ -q_1 v_3 + q_3 v_1 \\ q_1 v_2 - q_2 v_1 \end{bmatrix}, \tag{9}$$

and

$$\mathbf{q} \times \mathbf{v} \times \mathbf{q} = \begin{bmatrix} -q_1q_3v_3 + q_3^2v_1 - q_1q_2v_2 + q_2^2v_1 \\ -q_2q_3v_3 + q_3^2v_2 + q_1^2v_2 - q_1q_2v_1 \\ q_2^2v_3 - q_2q_3v_2 + q_1^2v_3 - q_1q_3v_1 \end{bmatrix}. \tag{10}$$

Plugging equations (8)–(10) into equation (7), we have the following equation:

$$\mathbf{R} = \begin{bmatrix} R_1(t) \\ R_3(t) \\ R_3(t) \end{bmatrix}, \tag{11}$$

where

$$\begin{aligned} R_1(t) &= v_1\left(q_0^2 + q_1^2 - q_2^2 - q_3^2\right) + v_2\left(-2q_0q_3 + 2q_1q_2\right) + v_3\left(2q_0q_2 + 2q_1q_3\right), \\ R_2(t) &= v_1\left(2q_0q_3 + 2q_1q_2\right) + v_2\left(q_0^2 - q_1^2 + q_2^2 - q_3^2\right) + v_3\left(-2q_0q_1 + 2q_2q_3\right), \\ R_3(t) &= v_1\left(-2q_0q_2 + 2q_1q_3\right) + v_2\left(2q_0q_1 + 2q_2q_3\right) + v_3\left(q_0^2 - q_1^2 - q_2^2 + q_3^2\right). \end{aligned}$$

Hence, the position of a vertex v_i can be represented by

$$\begin{aligned} p_i(t) &= T(t) + \mathbf{Q}(t)v_i\mathbf{Q}^{-1}(t) \\ &= \begin{bmatrix} T_1(t) \\ T_2(t) \\ T_3(t) \end{bmatrix} + \begin{bmatrix} R_1(t) \\ R_2(t) \\ R_3(t) \end{bmatrix}, \end{aligned}$$

where v_i is the relative position of V_i with respect to the centroid of the polyhedron P. Therefore, the distance function of $D_i(t)$ between V_i and the plane H (xy-plane) is the third component of $p_i(t)$,

$$\begin{aligned} D_i(t) &= T_3(t) + R_3(t) \\ &= T_3(t) + v_1\left(-2q_0q_2 + 2q_1q_3\right) + v_2\left(2q_0q_1 + 2q_2q_3\right) + v_3\left(q_0^2 - q_1^2 - q_2^2 + q_3^2\right). \end{aligned}$$

REFERENCES

1. J.E. Bobrow, A direct minimization approach for obtaining the distance between convex polyhedra, *The International Journal of Robotics Research* **8** (3), 65–76, (1989).
2. M.C. Lin and J.F. Canny, A new fast algorithm for collision detection, *Proceed. of Computer Animation*, 43–55, (1993).
3. F. Dai, Collision-free motion of an articulated kinematic chain in a dynamic environment, *IEEE Computer Graphics and Applications* **9** (1), 70–74, (1989).
4. M. Herman, Fast three-dimensional collision-free motion planning, *Proceed. of IEEE International Conference on Robotics and Automation* **2**, 1056–1063, (1986).
5. A. Pentland, Computational complexity versus simulated environment, *SIGGRAPH '93* **22** (2), 185–192, (1990).
6. H. Edelsbrunner, *Algorithms in Computational Geometry*, Springer-Verlag, (1987).
7. M.C. Lin, Efficient collision detection for animation and robotics, Ph.D. Thesis, Univ. of California, Berkeley, CA, (1993).
8. M.C. Lin and J.F. Canny, A fast algorithm for incremental distance calculation, *Proceed. of IEEE International Conference on Robotics and Automation*, 1008–1014, (1991).
9. M.K. Ponamgi, D. Manocha and M.C. Lin, Incremental algorithms for collision detection between polygonal models, *IEEE Transaction on Visualization and Computer Graphics* **3** (1), 51–64, (1997).
10. D. Baraff, Interactive simulation of solid rigid bodies, *IEEE Computer Graphics and Application*, 63–75, (May 1995).
11. M.P. do Carmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Englewood Cliffs, NJ, (1976).
12. H.S. Kim, Collision detection using spherical Voronoi diagrams, Ph.D. Thesis, KAIST, (1998).
13. H.S. Kim, H.O. Kim and S.Y. Shin, An efficient algorithm for determining the extreme vertices of a moving 3D convex with respect to a plane, *Computers Math. Applic.* **36** (3), 55–61, (1998).
14. H.S. Kim, A sequence of the extreme vertices of a moving regular polyhedron using spherical Voronoi diagram, *Journal of Korea Multimedia Society* **3** (3), 298–308, (2000).
15. H.S. Kim, A linear-time algorithm for computing the spherical Voronoi diagram of unit normal vectors of a convex polyhedron, *Journal of KISS: Computer Systems and Theory* **27** (10), 835–839, (2000).

16. R.E. Moore, *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ, (1966).
17. K. Mulmuley, *Computational Geometry: An Introduction through Randomized Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, (1994).
18. A. Aggarwal, L.J. Guibas, J.B. Saxe and P.W. Shor, A linear-time algorithm for computing the Voronoi diagram of a convex polygon, *Discrete & Computational Geometry* **4**, 591–604, (1989).
19. L.J. Guibas, J. Stolfi and K.L. Clarkson, Solving related two- and three-dimensional linear programming problems in logarithmic time, *Theoretical Computer Science* **49**, 81–84, (1987).
20. J.L. Junkins and J.D. Turner, Optimal continuous torque attitude maneuvers, *J. Guidance and Control* **3** (3), 210–217, (1980).
21. W.R. Hamilton, *Elements of Quaternions, Volume I, II*, Chelsea, (1969).
22. J.L. Junkins and J.D. Turner, *Optimal Spacecraft Rotational Maneuvers*, Elsevier, (1986).
23. M.J. Kim, M.S. Kim and S.Y. Shin, Orientation curves, *SIGGRAPH*, 111–122, (1995).